When is the Best Load Sharing Algorithm a Load Balancing Algorithm?

by

Phillip Krueger
Miron Livny

Computer Sciences Technical Report #694
April 1987

When is the Best Load Sharing Algorithm a Load Balancing Algorithm?

Phillip Krueger and Miron Livny

Computer Sciences Department
University of Wisconsin - Madison
Madison, WI 53706

April, 1987

Abstract

Two essentially different objectives have been identified for scheduling in distributed systems. While the objective of *load sharing* is to maximize the rate at which work is performed by the system, thus minimizing mean process wait time, the *load balancing* objective extends the concern of scheduling to *fairness*, minimizing, in addition to mean wait time, the mean and standard deviation of wait *ratio* and the correlation between wait ratio and CPU service demand. Correspondingly, the population of load distributing algorithms has been divided into those using the *load sharing strategy*, which assure that no node is idle while processes wait for service, and those following the *load balancing strategy*, which balance the workload among nodes. The goal of this paper is to examine the effects of the resource requirements of these strategies on their abilities to meet the load balancing and load sharing objectives. We find that the connection between the objectives and their respective strategies is weakened when these resource requirements are not negligible. While, under some conditions, each objective is best met by its corresponding strategy, under other conditions it is best met either by the alternative strategy or *without* load distributing. Furthermore, the factors that govern which strategy is best for a given objective are dynamic. We conclude that to be effective at meeting either objective over the wide range of conditions occurring within a distributed system, a load distributing algorithm must adapt to its environment.

1. Introduction

Scheduling in distributed systems is significantly more complex than for single-processor systems. A distributed scheduling policy for a general-purpose system can be divided into two components: a *local scheduling discipline* determines how the local resources at a single node are allocated among the resident processes, while a *load distributing strategy* distributes the workload among the nodes through process migration. Eager, et al. [Eager86] have noted that within the myriad load distributing algorithms proposed in the literature lie two distinct strategies for improving performance. *Load balancing* (LB) algorithms strive to equalize the workload among nodes, while *load sharing* (LS) algorithms simply attempt to assure that no node is idle while processes wait for service. In [Krueg86], it is shown that these strategies are associated with different performance objectives.

Load distributing requires the use of resources. Increased use increases contention, increasing queue delays for processes sharing those resources and decreasing the responsiveness of load distributing. Thus, in attempting to improve performance, load distributing incurs a performance cost. In this paper, we address the questions: How do the costs incurred by the LB and LS strategies affect the ability of each to meet its performance objective? Are there conditions under which the LS objective is *not* best met by the LS strategy, or the LB objective by the LB strategy? Based on this study, we identify guidelines for the design of new load distributing algorithms for general-purpose systems, to be explored in continuing research.

Improving performance is a difficult goal to define. The users of a computer system have certain performance expectations. The goal of a scheduling policy is to allocate resources in such a way that user expectations are most nearly met. User expectations can be embodied in a performance metric, which a scheduling policy attempts to optimize. Just as there is no universal set of user expectations, no single performance metric is applicable to every system. Different aspects of performance are measured by different performance indices, such as the means and standard deviations of process wait times and wait ratios, system throughput, and resource utilizations. These indices can be used singly or in combination to construct a performance metric matching the expectations of the users. When optimization of relevant performance indices is mutually exclusive, a performance metric must compromise, weighing gains in one against losses in another.

User performance expectations generally center on the quality of service provided to the processes they initiate. In addition to the average quality of service received, fairness is an important concern. Two users simultaneously initiating equivalent processes expect to receive about the same quality of service. Similarly, a user submitting the same job several times, under equivalent workloads, expects each to receive about the same quality of service. To ensure fairness, the variance in quality of service under a given workload should be acceptably low. To embody this aspect of fairness in a performance metric, the standard deviation of the quality-of-service metric, in addition to the mean, is an important performance index.

A second aspect of fairness relates to the way in which quality of service is measured. Both wait time and wait ratio are accepted measures of the quality of service received by a process. **Wait time** is the total amount of time a process spends waiting for resources, while **wait ratio** is the wait time per unit of service. Which measure is used carries an implied assumption about the importance of fairness. The use of wait time implies that the important factor in assessing quality of service is the absolute amount of time one waits for a resource, *regardless* of one's service demand. A person wanting to check out a book from a library and a person requesting a complete library tour would be considered to have received equal service if each waited the same amount of time for the attention of the librarian. The use of wait ratio implies that the important factor is the amount of time one waits for a resource *relative* to one's service demand. In providing equal quality of service, the person requesting an exhaustive library tour would be expected to wait longer for that service than the person wanting to borrow a book. The use of wait ratio, in preference to wait time, allows a more fair comparison of quality of service received.

Finally, fairness may imply that scheduling is *non-discriminatory*: variation in quality of service received by processes should be strictly random. The correlation between the quality-of-service metric and service demand can be used to measure an important aspect of discrimination in scheduling.

In [Krueg86], it is shown that when the resource requirements of load distributing are negligible, the LB and LS strategies are associated with different objectives. The LB strategy, in balancing the workload among nodes, provides approximately equal quality of service to all active processes, while the LS strategy maximizes the rate at which work is performed by the system by assuring that no node is idle while processes wait for ser-

vice. These objectives can be restated in terms of performance: While the objective of LS is simply to reduce total system wait time, thus reducing mean process wait time, the objective of LB is to reduce the mean and standard deviation of process wait *ratio* and the correlation between wait ratio and CPU service demand, as well as mean wait time. In this paper, we examine how the abilities of the LB and LS strategies to meet their performance objectives are affected by the performance cost incurred through their resource requirements. In the following pages, we show that:

- The cost incurred by the LB strategy is often greater than that of LS.
- The strategy that best meets the LB or LS objective varies with changes in workload characteristics and resource availability.
- Sometimes, improvement in one performance index is at the cost of another.
- The performance resulting from load distributing is very sensitive to the resource requirements of primitive load distributing operations, such as negotiation and migration.

We conclude that, while there is an essential distinction between the LB and LS objectives, the LB and LS strategies do not form a dichotomy, but are simply two points on a continuum of strategies that can be used to meet either the LB or LS objective. When the resource requirements of load distributing are considered, the strategy that best meets the LB or LS objective is not determined solely by the objective, but by the objective *together* with the system workload and resource availability. To be effective at meeting either the LB or the LS objective over the wide range of conditions occurring within a distributed system, a load distributing algorithm must adapt to its environment.

2. Distributed System Model

The models used in this study are closely related to the m*(M/M/1) family of distributed system models proposed by Livny [Livny83], but augmented to allow the processor queuing discipline, or *local scheduling discipline*, to be specified as a parameter. Figure 2.1 illustrates a system of this type, which consists of m processing elements, connected by a communication device. A load distributing algorithm allows processes to migrate between nodes at any time during their execution. Each of the m processing elements provides identical func-

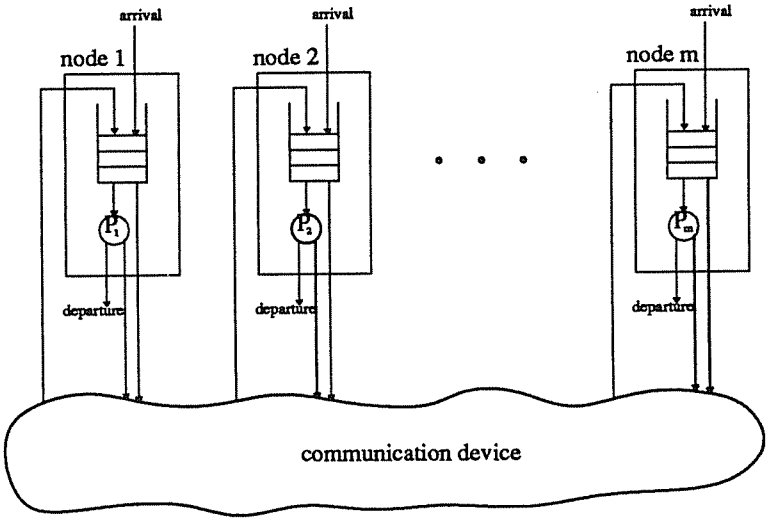

Figure 2.1 An m*(M/M/1) system

tional capabilities. Tasks execute equally well at any node, independent of the node where they arrived. Memory is not represented in the model. Tasks, having exponentially distributed service demands, arrive independently at each node and join the queue. The distribution of interarrival times is exponential, so the task arrival process of the entire system consists of m independent Poisson processes.

For this study, we assume an idealized communication device with a single queue using the First-Come-First-Served queuing discipline. Processes are assumed to execute independently, with no intercommunication. We consider only those models in which nodes have equal processing bandwidths and the service demands of processes arriving at different nodes are identically distributed. The rates at which processes initially arrive (as opposed to arriving as the result of a migration), however, may be different at different nodes. We refer to such a workload as having *inhomogeneous initiation rates*. Mutka and Livny [Mutka87] observed that, on a collection of workstations, user processes had executed within the previous 7 minutes on an average of only 30% of the nodes, varying with time of day from 18% to 53% of the nodes. Based on this data, we expect process initiation rates in some distributed systems to be generally inhomogeneous, with wide variability in the degree of inhomogeneity.

m	number of nodes comprising the distributed system
n	total number of processes present in the distributed system
p_n	probability that the number of processes in the system is n (see Laven83)
n_i	number of processes residing at node i
λ	arrival rate of processes to the system
λ_i	rate at which processes initiate at node i
μ_i	process service rate at node i
ρ_i	utilization of node i $= \lambda_i / \mu_i$
ρ	system load $= \sum_{i=1}^{m} \rho_i / m$
x	service demand of a process
$\bar{X}$	mean process service demand
div	integer division operator: y *div* z is equal to the truncated quotient of y/z
mod	modulus operator: y *mod* z is equal to the remainder of y *div* z

Table 2.1 Notation

Since we are interested in scheduling for general-purpose computer systems, we assume that the scheduler has no deterministic a priori information about process service demands. In addition, we assume that processes do not leave the system before completing service. The notation used throughout this paper is summerized in table 2.1.

A special case of the m*(M/M/1) model is the M/M/m-like system [Livny83]. Such a system simplifies analysis by making the assumption that no work is associated with scheduling: both context switches and process migrations are instantaneous and without cost. An M/M/m-like system is *work-conserving* [Klein76] if its distributed scheduling policy guarantees that no node is idle while processes wait for service, since such a policy, together with the above assumption, assure that "no work (service requirement) is created or destroyed within the system." While this study is centered on the more realistic m*(M/M/1) model, the M/M/m-like system is used in some analyses to establish bounds.

3. The Unshared and Unbalanced States

The performance cost incurred by load distributing in an m*(M/M/1) system, as well as the resulting performance gains, can be roughly gauged by how great an effect load distributing has on the distribution of load among nodes in an M/M/m-like system. Exerting greater influence implies a higher migration rate, requiring heavier use of resources and incurring greater cost. To measure this influence, we calculate the probability that the distribution of load must be modified either to conserve work or to balance the load. More specifically, we

calculate the probabilities that an M/M/m-like system without load distributing is in an unbalanced or unshared state. The system **unbalanced** state occurs when the processor queues on some pair of nodes differ in length by more than one, while the **unshared** state[1] is characterized by a node being idle while a process waits for service at another node. The set of unshared states is a subset of the set of unbalanced states. When $n \leq m+1$, a system that is shared is balanced as well, but when $n > m+1$, a shared system may be unbalanced.

Holding other factors constant, an M/M/m-like system without load distributing is more likely to be unbalanced or unshared when process initiation rates are inhomogeneous. The upper bound for the unbalanced and unshared probabilities occurs when the initiation rate at some node is zero, resulting in a system that is always both unbalanced and unshared. The lower bounds for each of these probabilities occur when process initiation rates are homogeneous. Under such a workload, this system can be decomposed into m independent and identical M/M/1 queues. Livny [Livny82] has shown that the probability that such a system is unshared is:[2]

$$P_L(unshared) = 1 - \rho^m(1-(1-\rho)^m) - (1-\rho^2)^m$$

For such a system to be balanced, there must exist some k such that i of the m nodes have queue length k, while the remaining $m-i$ nodes have queue length $k+1$. The probability that this system is unbalanced is then:[2]

$$P_L(unbalanced) = 1 - \sum_{k=0}^{\infty}\sum_{i=1}^{m}\binom{m}{i}\left[(1-\rho)\rho^k\right]^i\left[(1-\rho)\rho^{k+1}\right]^{m-i} = 1 - \frac{(1-\rho^2)^m - (\rho-\rho^2)^m}{1-\rho^m}$$

These probabilities are displayed in fig. 3.1. For reasonably sized systems and typical system loads, a distributed system without load distributing is generally in an unbalanced or unshared state, even when nodes and process initiation rates are homogeneous. Even at this lower bound, load distributing exerts an almost continuous influence on the distribution of load among nodes. While the probability that this system is unbalanced rises monotonically with system load, the probability that it is unshared falls off at high system loads. The migration rates required for the LB and LS strategies under such a workload can be expected to follow these tendencies. When nodes and workloads are homogeneous, the activity of an efficient LS algorithm should decrease at high system loads (as noted in [Livny82]) while that of an LB algorithm should not. Since this

1. Livny [Livny82, Livny83] refers to this as the Wait while Idle (WI) state.

2. These results also hold when service demands are generally distributed, if the queuing discipline is Processor Sharing.

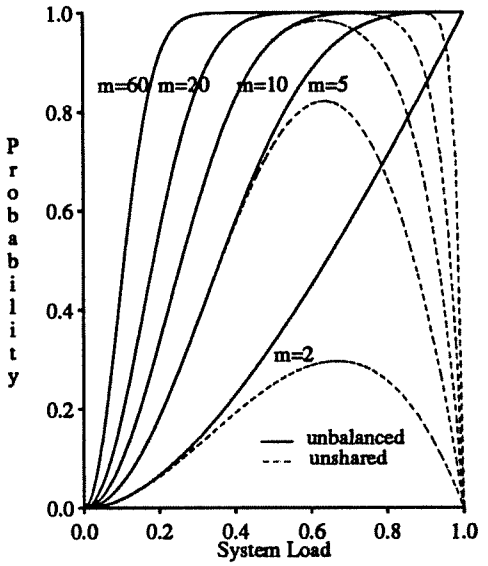

Figure 3.1 Probabilities of unshared and unbalanced states

activity requires the use of resources, we can expect that at a sufficiently high system load, the resource contention caused by LB, with resulting performance cost, will be greater than the potential improvement in performance. At such a load, the LB strategy will be ineffective at meeting its objective; the LB objective will be best met with a reduced activity level.

While calculating the probabilities of the unbalanced and unshared states provides insight into the relative migration rates of the LB and LS strategies when initiation rates are homogeneous, it gives no insight when the initiation rate at some subset of nodes is zero. Under such a workload, both probabilities are 1. To roughly gauge the difference in migration rates over the range of inhomogeneities in initiation rates, we calculate the probability that the system is unbalanced given that it is kept in a shared state. The upper bound for this probability occurs when all processes initiate at the same node. In this case, when there are more than m processes in the system, all $n-m$ processes that remain after each node has been allocated one will remain at their initiation node. The upper bound for the probability that an M/M/m-like system is unbalanced given that it is shared is:

$$P_U(unbalanced \mid shared) = \sum_{n=m+2}^{\infty} p_n \tag{3.1}$$

The lower bound for this probability occurs when initiation rates are homogeneous. In this case, when $n > m$, each of the $n-m$ 'excess' processes remaining after one has been allocated to each node can reside on

any node. The total number of ways in which these $n-m$ processes can be distributed among the m nodes is m^{n-m}. As an approximation, we assume that each of these configurations is equally likely. If the load is balanced, each of the $j = n \ mod \ m$ nodes have $i+1 = n \ div \ m$ of the $n-m$ excess processes, while the remaining $m-j$ nodes each have i processes. The number of configurations that cause the system to be balanced is:

$$b_n = \begin{cases} \prod_{k=0}^{m-1} \binom{(n-m)-ki}{i} & \text{for } j = 0 \\ \binom{m}{j} \left[\prod_{k=0}^{j-1} \binom{(n-m)-k(i+1)}{i+1} \right] \left[\prod_{k=0}^{m-j-1} \binom{(n-m)-j(i+1)-ki}{i} \right] & \text{for } j = 1,2,\ldots,m \end{cases}$$

An approximate lower bound is then:

$$P_L(unbalanced \mid shared) = \sum_{n=m+1}^{\infty} (1 - (b_n / m^{n-m}))\, p_n \qquad (3.2)$$

Simulation results show that there is a tendency toward configurations that are unbalanced. For example, when $n-m = 4$, it is more likely than predicted that all 4 of these 'excess' processes reside on one node, while being less likely than predicted that 2 reside on each of two nodes. If the actual distribution of these $n-m$ processes among the m nodes were known, this tendency would allow a larger value to be found for $P_L(unbalanced \mid shared)$. Thus, eq. 3.2 is a valid lower bound.

Probabilities resulting from eqs. 3.1 and 3.2 have been calculated[1] and are displayed in fig. 3.2. As the system load or inhomogeneity in process initiation rates increases, or the number of nodes decreases, the probability that an M/M/m-like system using load sharing is unbalanced increases. This is intuitively reasonable, since for $n > m$, $\lim_{m \to \infty} p_n = 0$ and $\lim_{\rho \to 0} p_n = 0$. Under opposite conditions, as the probability that the system is unbalanced decreases, an efficient LB algorithm should degrade to the LS strategy. If an LB algorithm requires much overhead to determine whether the load is balanced, it will not be efficient under such conditions. The overhead of a suitable LB algorithm should be greater than that for an LS algorithm only in proportion to the additional process migrations that are necessary.

1. Infinite summations are carried through all increments having values greater than 0.0001.

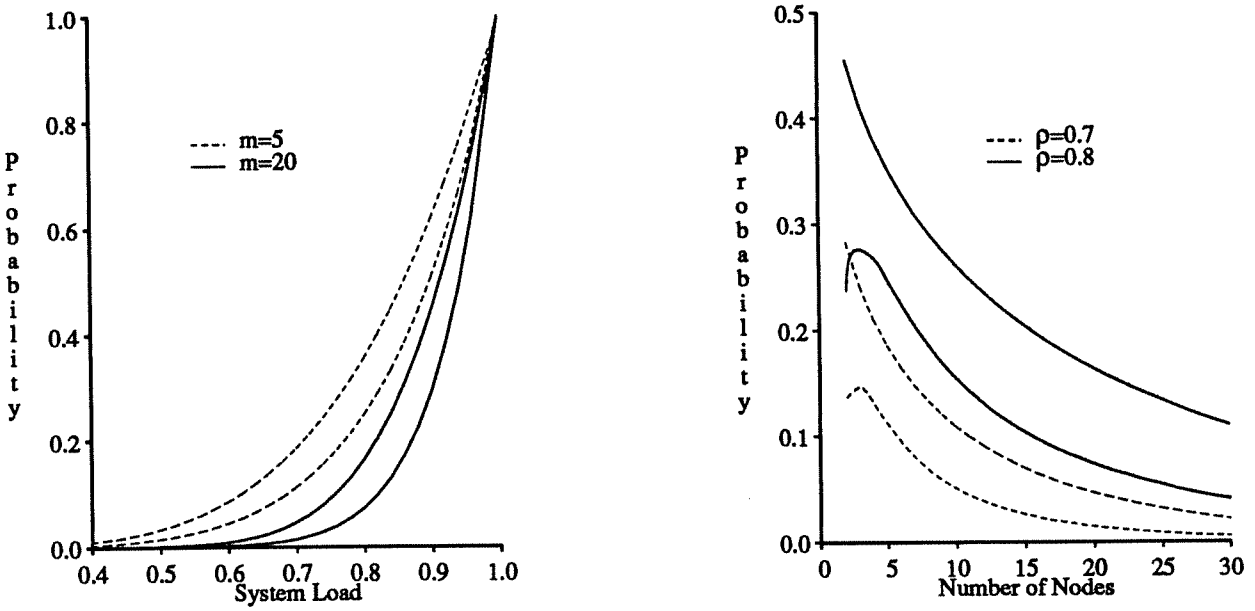

Figure 3.2 Upper and lower boundaries for the probability that an M/M/m-like system is unbalanced given that it is shared vs. system load (left) and number of nodes (right)

4. Negotiation, Migration and Unshared Rates and the Blurred Distinction Between LB and LS

In addition to process migrations, load distributing activity generally includes negotiations among nodes for those migrations. A negotiation session entails a node searching for a *migration partner*, a complementary node that can either accept or provide a migrating process. As discussed in [Krueg86], the objectives of negotiation and migration activity differ for LB and LS. LB algorithms attempt to provide equal quality of service to all active processes, while LS algorithms simply attempt to maximize the rate at which work is performed by the system by avoiding unshared states, thus minimizing mean wait time. While the distinction between the strategies used to approach these objectives is clear when the resource requirements of negotiation and migration are negligible, it blurs as these resource requirements become increasingly significant. If negotiation and migration are instantaneous, there is no advantage, from the point of view of the LS objective, in migrating processes to nodes that are not idle. However, when negotiation and migration require time to complete, an idle node is unable to immediately acquire a new process, and *anticipatory* migrations [Livny83], migrations to nodes that are not idle but are expected to soon become idle, may aid in avoiding unshared states. From the point of view of the LS objective, the LB strategy is an extreme: it performs the greatest number of useful anti-

cipatory migrations. Alternatively, from the point of view of the LB objective, as the LS strategy includes more anticipatory migrations, it becomes a closer approximation of the LB strategy.

Our goals in this section are two-fold. Initially, we wish to identify the system parameters that determine the negotiation and migration rates resulting from LB and LS. In addition, we wish to examine the effects of variations in these parameters on those rates, paying particular attention to the level of activity beyond that of LS required for LB. A second goal, considering LB from the point of view of the *LS* objective, is to examine the usefulness of the additional migrations, which are considered anticipatory in this framework, in reducing the rate at which individual nodes enter the unshared state, referred to as the *unshared rate*. We derive these rates[1] for an M/M/m-like system. By comparing the activity level of LB beyond that of LS with the usefulness of its additional migrations as anticipatory migrations, we can predict the conditions under which LB most favorably compares with LS in meeting the LS objective.

Holding other parameters constant, the lower bounds for the negotiation, migration and unshared rates occur when process initiation rates are homogeneous across nodes, while the upper bounds are reached when all processes initiate at a single node. Since the derivations of these bounds are somewhat lengthy, they have been relegated to appendix A. While all the rates studied increase linearly with m and μ, figures 4.1 through 4.3 show that their relationship with λ is more complex. When process initiation rates are homogeneous across nodes, the LS negotiation and migration rates reflect the probability of the system unshared state displayed in figure 3.1, dropping off at high system loads. The unshared rate under LS has the same tendency. However, when all processes initiate at a single node, these rates increase monotonically with system load. Predicted by the probability of the system unbalanced state, the negotiation and migration rates of LB increase monotonically with system load regardless of the homogeneity in initiation rates. However, the unshared rate under LB behaves differently, falling off at high system loads. While LB has generally higher negotiation and migration rates than LS, it has a lower unshared rate.

1. In the interest of generality, rather than calculate the rates at which individual negotiation messages are generated, we calculate the rates at which negotiation *sessions* are initiated. This measure allows the negotiation rates in cost-free systems to be compared with the broadest range of load distributing algorithms for systems in which cost is incurred.

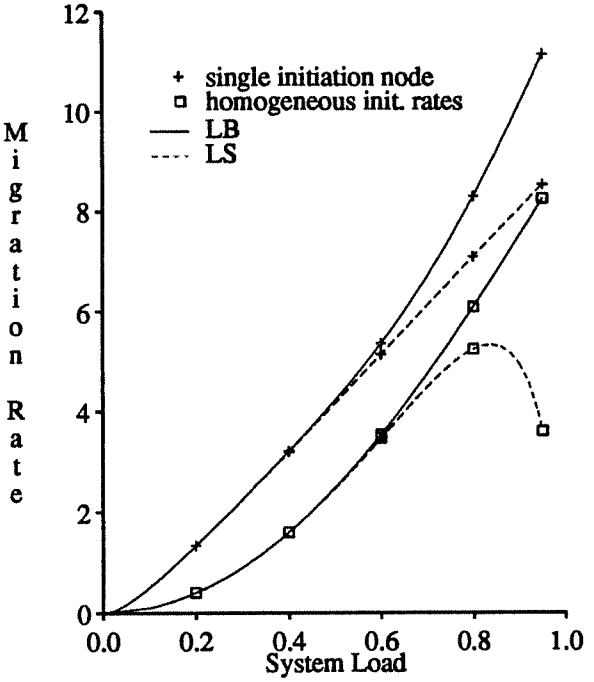

Figure 4.1 Migration rate vs. system load (varying λ with fixed μ) ($m = 10, \mu = 1$)

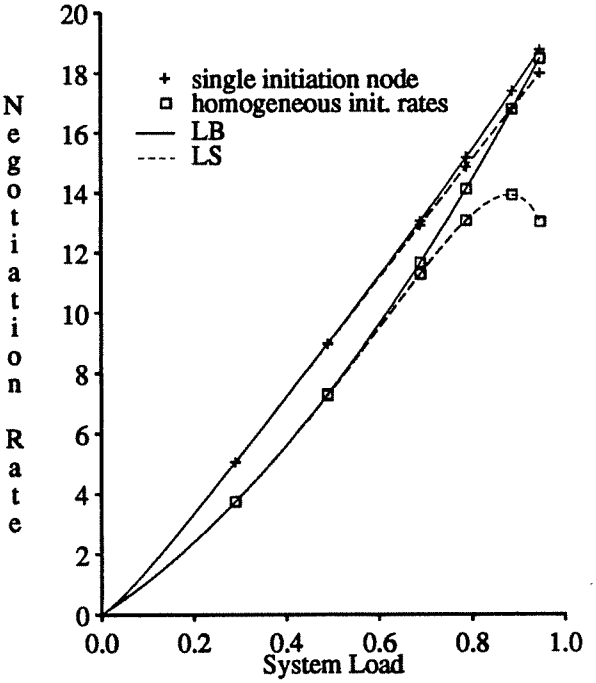

Figure 4.2 Negotiation rate vs. system load (varying λ with fixed μ) ($m = 10, \mu = 1$)

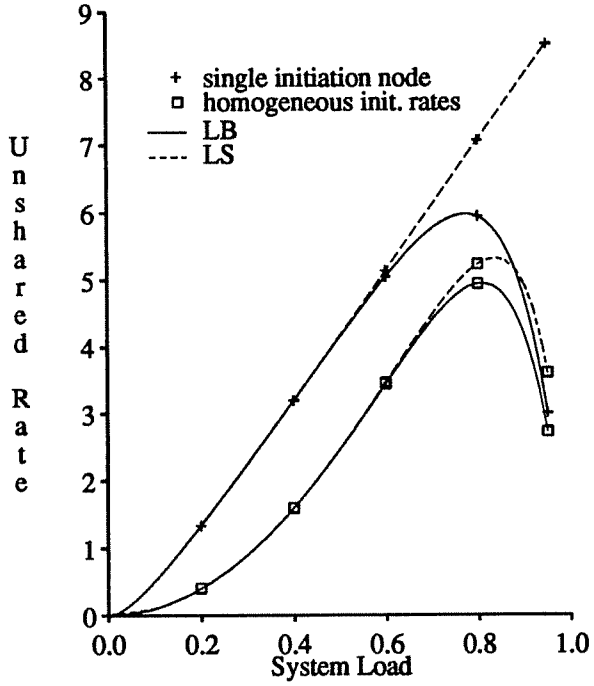

Figure 4.3 Unshared rate vs. system load (varying λ with fixed μ) ($m = 10, \mu = 1$)

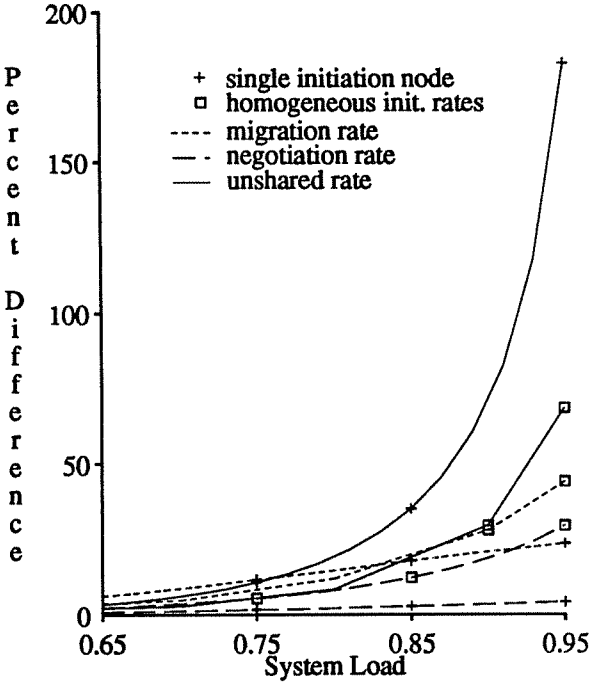

Figure 4.4 Absolute percent difference ($100 * |LB - LS| / LB$) in negotiation, migration and unshared rates vs. system load (m=10, μ=1)

Holding other parameters constant, figure 4.4 shows that, for both the negotiation and migration rates, the relative difference between LB and LS is least when all processes initiate at a single node. In contrast, the relative difference in unshared rates is greatest when there is a single initiation node. The condition resulting in the least difference in activity level also results in the greatest difference in ability to avoid node unshared states. When negotiation and migration require significant time to complete, LB can be expected to compare more favorably with LS in terms of the LS objective, avoiding unshared states, as the homogeneity in process initiation rates decreases. This effect will be more pronounced as the time required to complete a negotiation session and process migration increases, increasing the performance degradation caused by unshared states. In contrast, LB can be expected to compare least favorably with LS when initiation rates are homogeneous. This effect will be most pronounced at high system load.

5. Simulation Study of Systems with Significant Load Distributing Resource Requirements

As discussed in the introduction, the LB and LS objectives can be stated in terms of performance: The LS objective is to minimize $\overline{WT}$ (see table 5.1 for notation), while the LB objective is to minimize $\overline{WR}$, σ_{WR}, and $\rho(wait\ ratio, X)$, as well. In [Krueg86], it was shown that when the resource requirements of load distributing are negligible, the LB strategy is significantly better at meeting the LB objective than the LS strategy. Compared with LS, the LB strategy reduces $\overline{WR}$, and σ_{WR}, while maintaining the same $\overline{WT}$. When the LB strategy is extended to include *shuffling*, migrations between nodes that differ in load by one, $\rho(wait\ ratio, X)$ is significantly reduced, as well. When process service demands are hyperexponentially distributed, the performance advantage of the LB strategy extends to the *LS* objective: $\overline{WT}$ is reduced relative to that of the LS strategy. These potential performance advantages, which we refer to collectively as the *LB advantage*, were shown to increase with increasing inhomogeneity in process initiation rates, system load and coefficient of variation in

σ_{WR}	standard deviation of wait ratio
$\overline{WR}$	mean process wait ratio
$\rho(wait\ ratio, x)$	correlation between wait ratios and service demands
$\overline{WT}$	mean process wait time

Table 5.1 Notation for performance indices

process service demands. However, the previous sections show that both strategies require significant levels of activity, implying that the resource requirements of load distributing are not likely to be negligible. The primary resources used to perform process migrations and to pursue negotiations among nodes for those migrations are CPU processing bandwidth and communication device bandwidth. Increased use of resources increases contention, increasing queue delays for processes sharing these resources and decreasing the responsiveness of load distributing. Together, these factors result in a performance cost that offsets the potential performance gains of load distributing. The previous sections contained evidence that, as a result of this performance cost, LS may not always be the best strategy to meet the LS objective, nor LB for the LB objective. In this section, we substantiate these results through simulation.

Non-negligible load distributing resource demands complicate the design of load distributing algorithms. For example, when negotiation and migration require time to complete, the length of the CPU queue at a node is not a sufficient measure of its load. Since migrations are not instantaneous, the queue length does not change as soon as a migration is negotiated. However, a node must not commit itself to accept so many processes that, when they finally arrive, it becomes overloaded, since processor thrashing might result [Bryan81]. A load metric that avoids this problem augments the node's queue length with its *reservations* [Livny83], the change in queue length expected to be induced by processes that are in transit. The number of reservations recognized by a node increases when the node agrees to accept a migrating process, and decreases when the migrating process arrives or the node becomes convinced that the migrating process will never arrive. Another complication of non-negligible resource requirements is that the cost incurred by negotiating with every node in the system to find the best migration partner may be prohibitive. To maximize performance, a load distributing algorithm may be forced to limit negotiation to a subset of nodes. Finally, when these resource requirements are not negligible, sophisticated criteria for selecting a process to migrate become advantageous. In our experiments, the process selected to migrate:

(1) has the smallest migration size, which includes the size of its executable image as well as any data that must accompany it, among the processes residing at the overloaded node,

(2) belongs to the set of processes that have migrated the least often among those residing at the node,

(3) has executed for at least F * *CPU time required to migrate process*, where F is a parameter.

Unfortunately, in contrast to load distributing in cost-free (M/M/m-like) systems, many other design details affect performance. Different algorithms have different resource requirements and differ in responsiveness. For example, an algorithm using broadcast messages for negotiation may require less bandwidth of the communication device, but more CPU bandwidth, than a polling algorithm. In addition, a broadcast algorithm may require less time to negotiate a migration, and may thus be more responsive than a polling algorithm. Differences in resource use result in different patterns of resource contention, which in turn affect negotiation and migration rates, responsiveness, and interference with processes sharing those resources, thus affecting performance.

To compare the LB and LS strategies, we propose a single generalized polling algorithm, called *PollGen*, that takes on the characteristics of either the LB or LS strategy, depending on the values of its parameters. The PollGen algorithm is *symmetrically initiated*: a node initiates negotiation, searching for a suitable *migration partner*, on becoming either *overloaded* or *underloaded*. A node is considered overloaded or underloaded according to the relationship of its load to two static thresholds: A node becomes overloaded when the initiation of a process causes the load to be $\geq T_1$, and becomes underloaded whenever the completion of a process causes the node to become idle. Similarly, identification of a suitable migration partner depends on two static thresholds: A suitable partner differs in load from the node initiating negotiation by $\geq T_2$. Furthermore, when the initiator of a negotiation session is an overloaded node, the load of a suitable partner is $\leq T_3$.

In our experiments, the *flavor* of the algorithm, LB or LS, is controlled by T_3. Both T_1 and T_2 are always set to 2. An LS algorithm that performs no anticipatory migrations has $T_3 = 0$, while an algorithm that allows anticipatory migrations has $T_3 > 0$. To implement the LB strategy, T_3 is set to infinity. We refer to these different instances of the algorithm as *PollGen* (T_3).

Under the PollGen algorithm, the way in which negotiation proceeds depends on whether it is initiated by an idle node or an overloaded node. An idle node chooses the *first* suitable migration partner, so that it is not idle any longer than necessary. An overloaded node, however, searches for the *best* partner, an idle node.

Migrations that correct unshared states, then, take precedence over anticipatory or load balancing migrations. Through this mechanism, the load balancing algorithm, PollGen(∞), is able to degrade to load sharing, with very little additional overhead, when load balancing is unnecessary. To negotiate, an idle node polls a random set of *PollLimit* nodes until a suitable partner is found. If no suitable partner is found, the node remains idle. An overloaded node poll a random set of *PollLimit* nodes until an idle node is found. If no idle node is found, each suitable partner that has been found in this set is polled again, beginning with the node having the lowest load when last polled, until a partner that remains suitable is found. If no suitable partner is found, the node remains overloaded.

Since LB and LS, when allied with First-Come-First-Served as a local scheduling discipline, have been shown to be ineffective in reducing $\overline{WR}$ and σ_{WR} [Krueg86], we assume that PollGen is used in conjunction with Processor Sharing as a local scheduling discipline.

Results

We examine the performance of the PollGen algorithm with values for T_3 varying from LS without anticipatory migrations at one extreme (T_3=0), through LS with anticipatory migrations (T_3=1), to LB at the opposite extreme (T_3=∞). The analytic models presented by Eager, et al. [Eager86, Eager86a] are not applicable to this study because we wish to consider workloads having inhomogeneous process initiation rates and resource requirements for process migrations that are independent of service demands, in addition to examining the broad range of performance indices on which the LB objective is based. Instead, we rely on simulation, with all data presented having less than 6% error at the 95% confidence level.

We assume that the physical sizes of processes are independent of all other process characteristics and are exponentially distributed, as are process service demands and interarrival times. We also assume that the CPU service demands of negotiation and migration messages preempt all other processing. This assumption implies that processes may not begin to receive service immediately on initiating. In [Krueg86], it is shown that for such policies, $\overline{WR}$ and σ_{WR} are infinite when measured over the entire population of processes. To avoid this problem, processes having the shortest 1% of service demands are trimmed from the sample when measuring these performance indices. Some comparisons between performance indices resulting from PollGen are

plotted as: $Percent\ Difference = 100\ ((PollGen(T_3) - PollGen(\infty))\ /\ PollGen(\infty))$. All $\overline{WT}$ measurements are normalized by $\bar{X}$. Default parameters are:

System load (ρ)	0.8
Mean process CPU service demand ($\bar{X}$)	20 time units
Mean process size ($\bar{S}$)	120K bytes = 960K bits
CPU service demand for migration packets	.004 time units
Maximum packet size	4K bytes
CPU service demand for negotiation messages	.002 time units
Negotiation message size	32 bytes
Communication device bandwidth	10 million bits/time unit
PollLimit	5
F	0.1

The CPU requirement incurred by a migrating process of average size is then 0.6% of $\bar{X}$ at both the sending and receiving node, for a total of 1.2%. Each poll requires 0.02% of $\bar{X}$ at each node for a total of 0.04%.

Figures 5.1 through 5.4 plot $\overline{WT}$, $\overline{WR}$ and σ_{WR} against the CPU performance cost incurred by load distributing, while figures 5.5 through 5.7 plot each index against communication device cost. For this study, CPU cost is manipulated by varying the total CPU requirement for migrating a process of average size, and communication device cost is manipulated by varying the bandwidth of the communication device. While the communication device bandwidth and the CPU overhead of individual messages are generally static, variations in dynamic characteristics of the workload or resource availability have effects equivalent to variations in these parameters. For example, variation in the utilization of the communication device for purposes other than load distributing affects communication device cost. Variation in the number of nodes participating in the system, which may occur in systems composed of workstations or as a result of node failure, affects the total negotiation and migration rates, changing communication device cost. Such variation may also affect the rate at which nodes receive negotiation messages, affecting CPU cost. Another example, variation in the distribution of physical sizes of processes, changes both the CPU and communication device costs incurred by migrating processes. Equivalently, variation in the distribution of process service demands (at constant system load) affects the negotiation and migration rates, changing the overall CPU and communication device costs incurred. To underscore the relationship between process size, service demand and the CPU requirement of load distributing, CPU cost is plotted as the percent of $\bar{X}$ required to migrate a process of average size.

Figures 5.1 through 5.7 show that the effect that CPU and communication device cost have on the LB performance advantage depends on the pattern of process initiation rates. When there is a single initiation node,[1] the performance advantage of PollGen(∞) relative to PollGen(0) *improves* with increasing cost. For all non-negligible costs, PollGen(∞) best meets the LB objective. Additionally, consistent with evidence in the previous section, non-negligible cost causes degradation in $\overline{WT}$ due to node unshared states, allowing PollGen(∞) to best meet the *LS* objective by significantly reducing the unshared rate. In contrast, when initiation rates are homogeneous, increasing cost *degrades* the PollGen(∞) performance advantage. PollGen(∞) best meets the LB objective only at low cost. In terms of the LS objective, the results again follow evidence from the previous section: PollGen(0) best meets the LS objective under such a workload, except at very high levels of cost. The relatively high negotiation and migration rates of PollGen(∞) can not be justified by the small decrease in the node unshared rate. The lower level of anticipatory migrations generated by PollGen(1) also results in poorer $\overline{WT}$, showing that anticipatory migrations are not justified under such a workload.

While PollGen(∞) best meets both the LB and LS objectives when processes initiate at a single node, selecting the best instance of PollGen is not as obvious when initiation rates are homogeneous. Figures 5.3 and 5.4 shows that when CPU cost is low, PollGen(∞) pays for improved $\overline{WR}$ and σ_{WR} with poorer $\overline{WT}$ than the other instances of PollGen. At increased levels of CPU cost, improved σ_{WR} is at the expense of both $\overline{WR}$ and $\overline{WT}$. At still higher cost, PollGen(∞) does not meet any part of the LB objective as well as PollGen(0), which also best meets the LS objective. Finally, when CPU cost is very high, the performance of a system *without* load distributing, in terms of either objective, is better than that of any instance of PollGen. At this level of CPU cost, even the relatively modest cost incurred by PollGen(0) outweighs any potential improvement in performance. At intermediate levels of cost, PollGen(1) is a contender for best meeting the LB objective. However, in this region, selection of the best algorithm depends on the relative weights given $\overline{WR}$, σ_{WR} and $\overline{WT}$, since each trades improvement in one set of indices for degradation in another.

When initiation rates are homogeneous, figure 5.10 shows that another transition in the instance of PollGen that best meets the LB or LS objective occurs with variations in system load, as predicted in the

1. Results for PollGen(0) under such a workload are not plotted for m=20, since PollGen(0) is unstable under such conditions.

previous section. For m=20, with the total CPU requirement to migrate a process of average size = 2.4%, the instance of PollGen that minimizes each performance index is:

System Load	$\overline{WR}$	σ_{WR}	$\overline{WT}$
0.5	any	any	any
0.7	any	1, ∞	any
0.8	∞	∞	∞
0.9	1	∞	0, 1
0.95	1	1	0, 1

Varying levels of inhomogeneity can also result in transitions in the instance of PollGen that best meets the LB or LS objective. The transition from a workload in which both the LB and the LS objectives are best met by PollGen(∞) to one in which both objectives are most nearly met by PollGen(0) is plotted in figures 5.8 and 5.9. At intermediate levels of inhomogeneity, PollGen(1) best meets the LS objective and, depending on the relative weights accorded $\overline{WR}$, σ_{WR} and $\overline{WT}$, may most nearly meet the LB objective, as well.

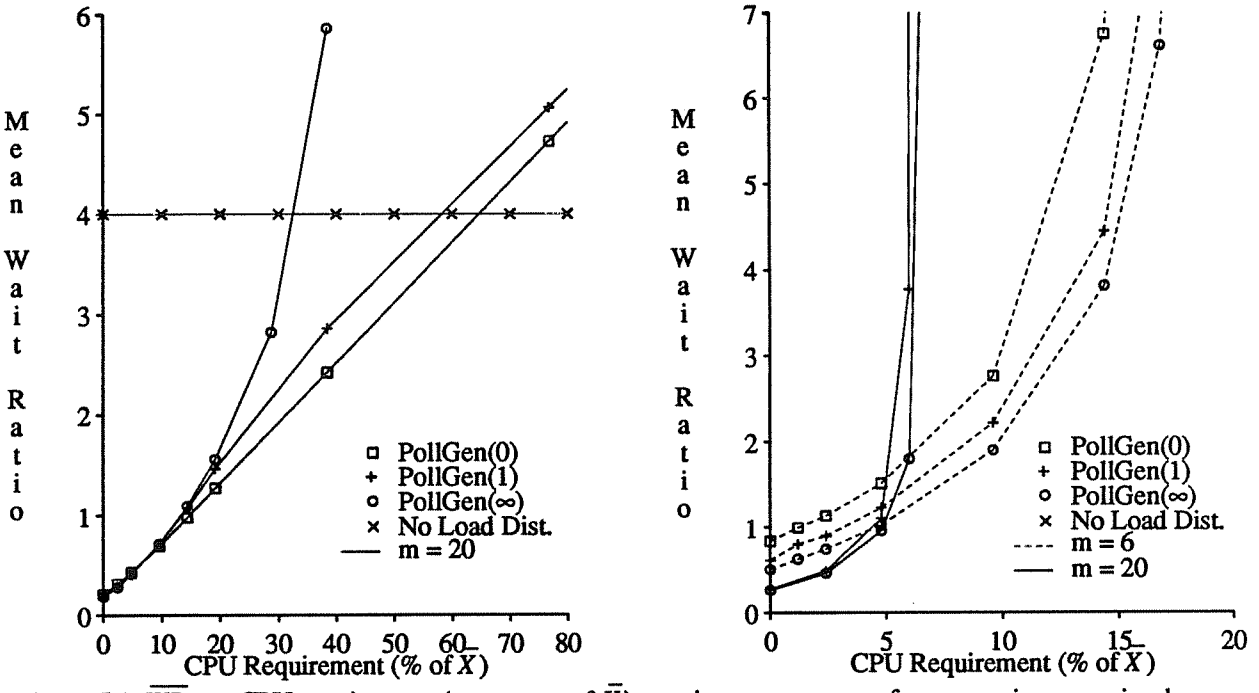

Figure 5.1 $\overline{WR}$ vs. CPU requirement (as percent of $\bar{X}$) to migrate a process of average size assuming homogeneous initiation rates (left) or all initiations at a single node (right) ($\rho = 0.8$)

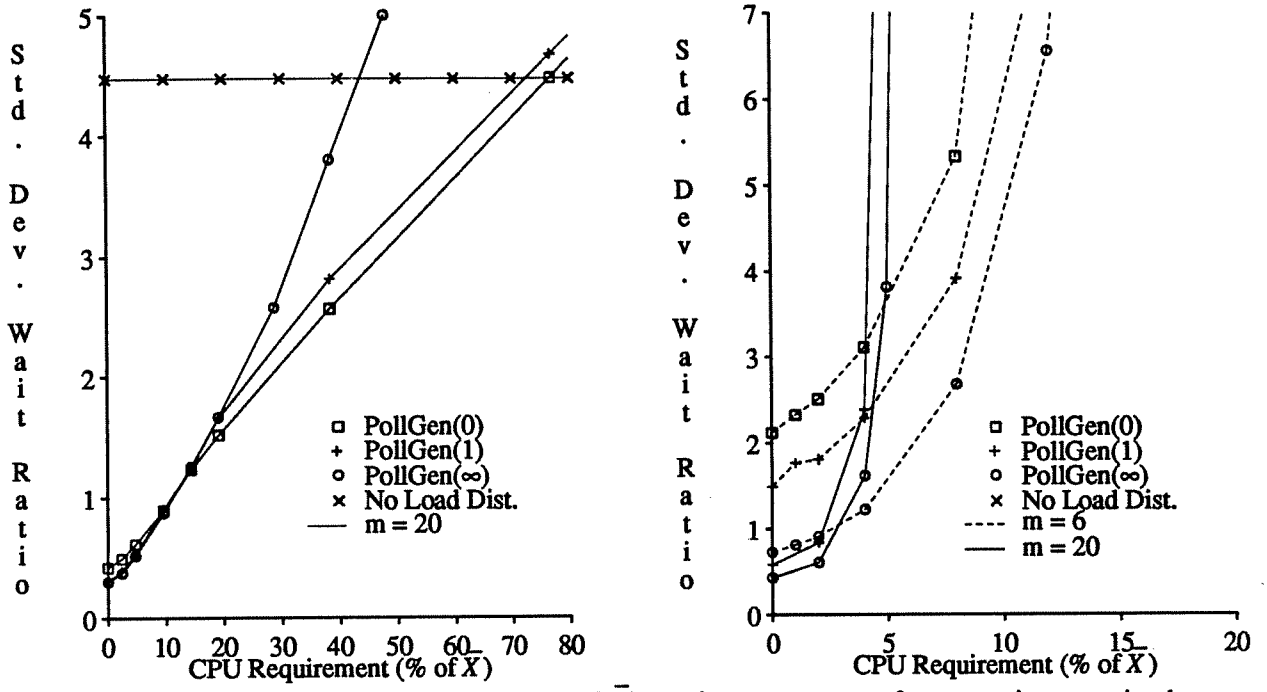

Figure 5.2 σ_{WR} vs. CPU requirement (as percent of $\bar{X}$) to migrate a process of average size assuming homogeneous initiation rates (left) or all initiations at a single node (right) ($\rho = 0.8$)

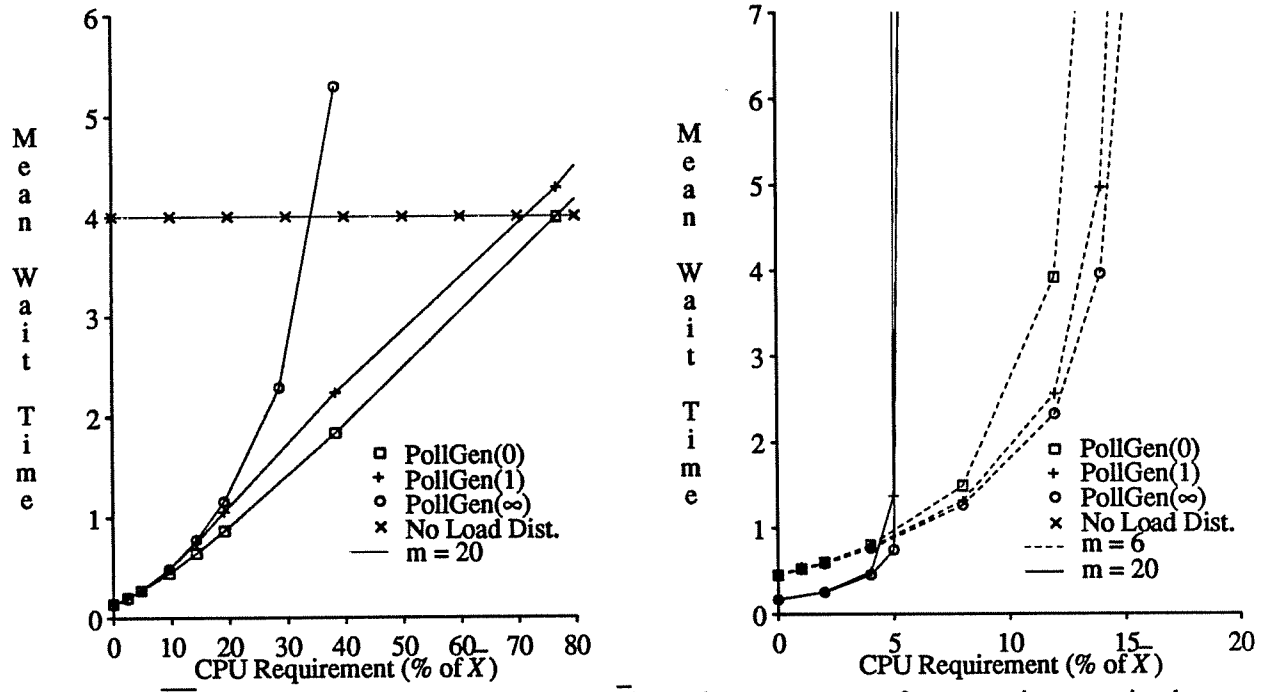

Figure 5.3 $\overline{WT}$ vs. CPU requirement (as percent of $\bar{X}$) to migrate a process of average size assuming homogeneous initiation rates (left) or all initiations at a single node (right) ($\rho = 0.8$)

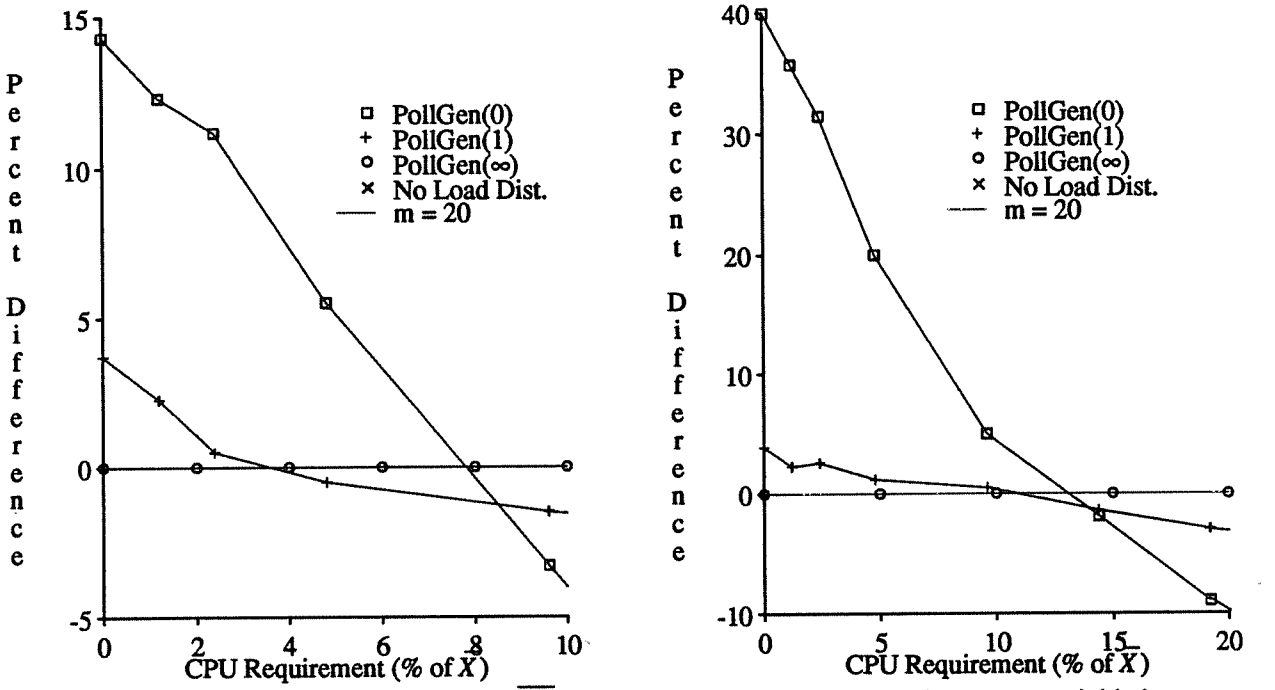

Figure 5.4 Closeup of percent difference in $\overline{WR}$ (left) and σ_{WR} (right) assuming *homogeneous* initiation rates vs. CPU requirement (as percent of $\bar{X}$) to migrate a process of average size ($\rho = 0.8$, $m = 20$)

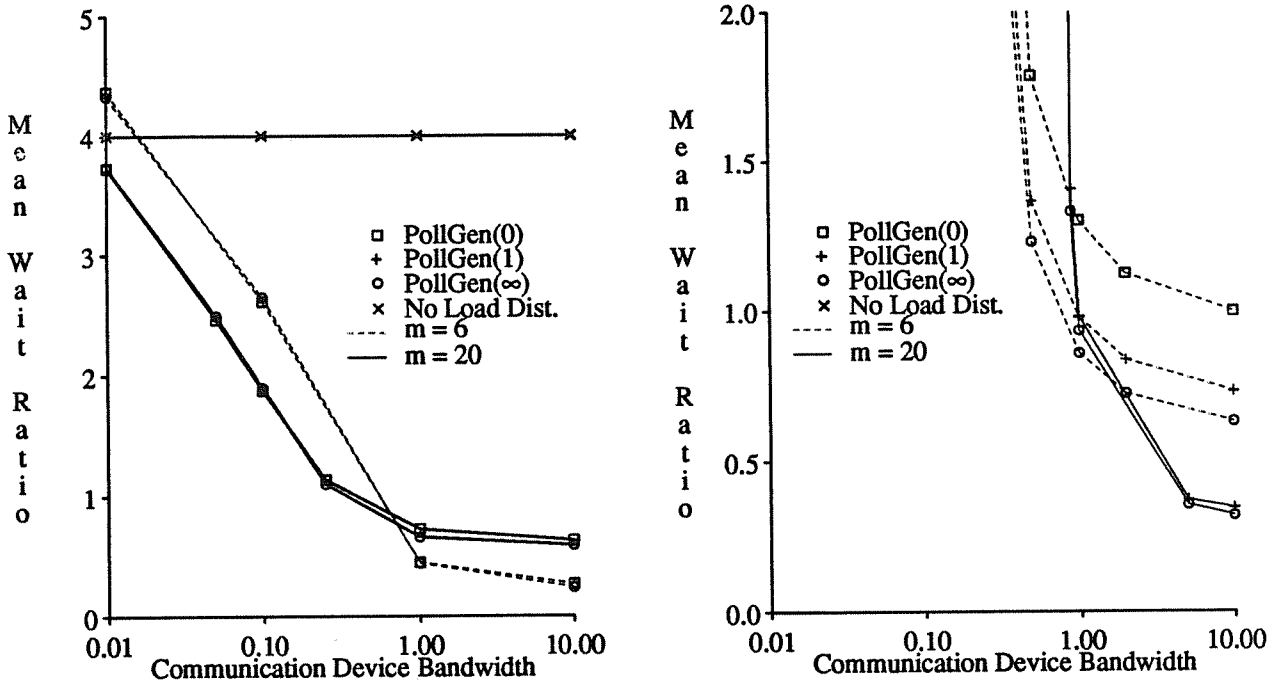

Figure 5.5 $\overline{WR}$ vs. communication device bandwidth (Mbits / time unit) assuming homogeneous initiation rates (left) or all initiations at a single node (right) ($\rho = 0.8$)

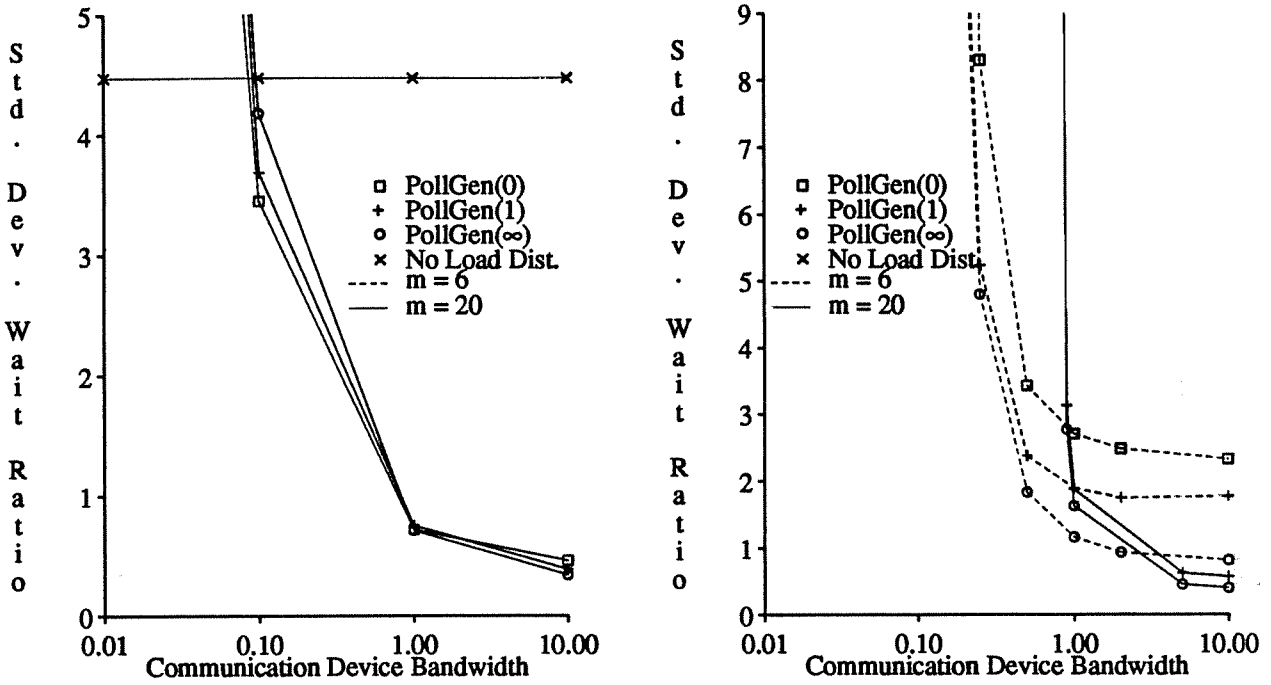

Figure 5.6 σ_{WR} vs. communication device bandwidth (Mbits / time unit) assuming homogeneous initiation rates (left) or all initiations at a single node (right) ($\rho = 0.8$)

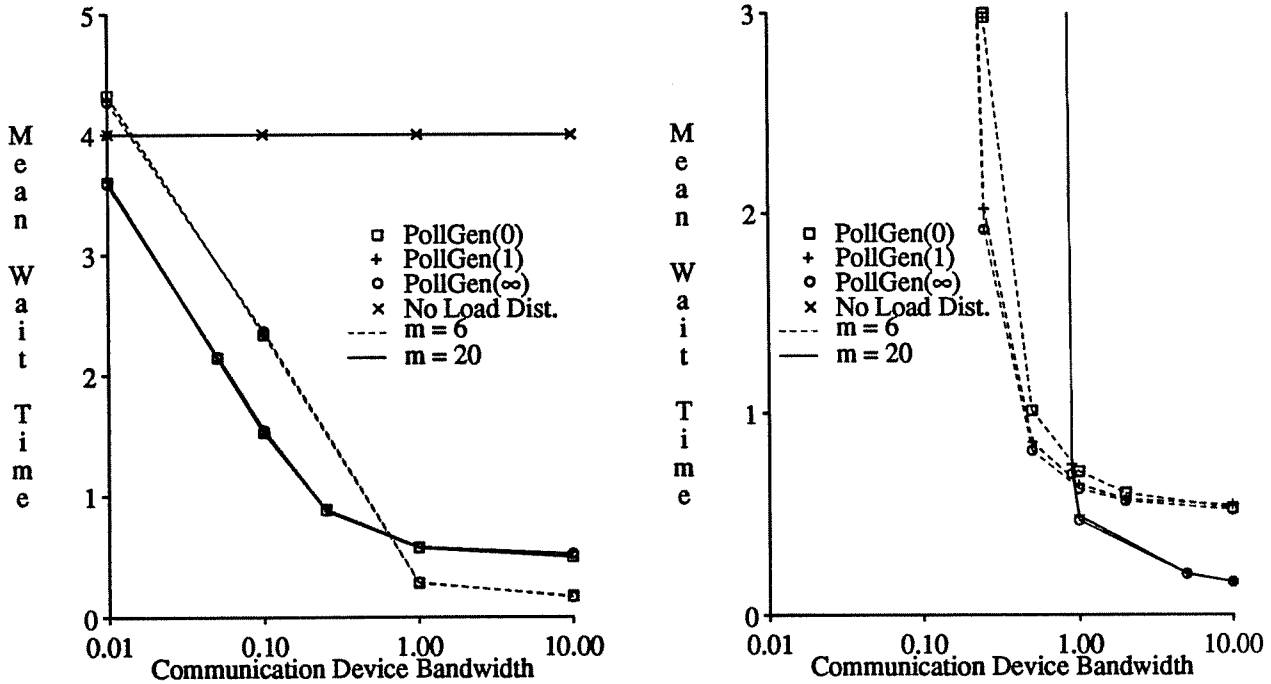

Figure 5.7 $\overline{WT}$ vs. communication device bandwidth (Mbits / time unit) assuming homogeneous initiation rates (left) or all initiations at a single node (right) ($\rho = 0.8$)

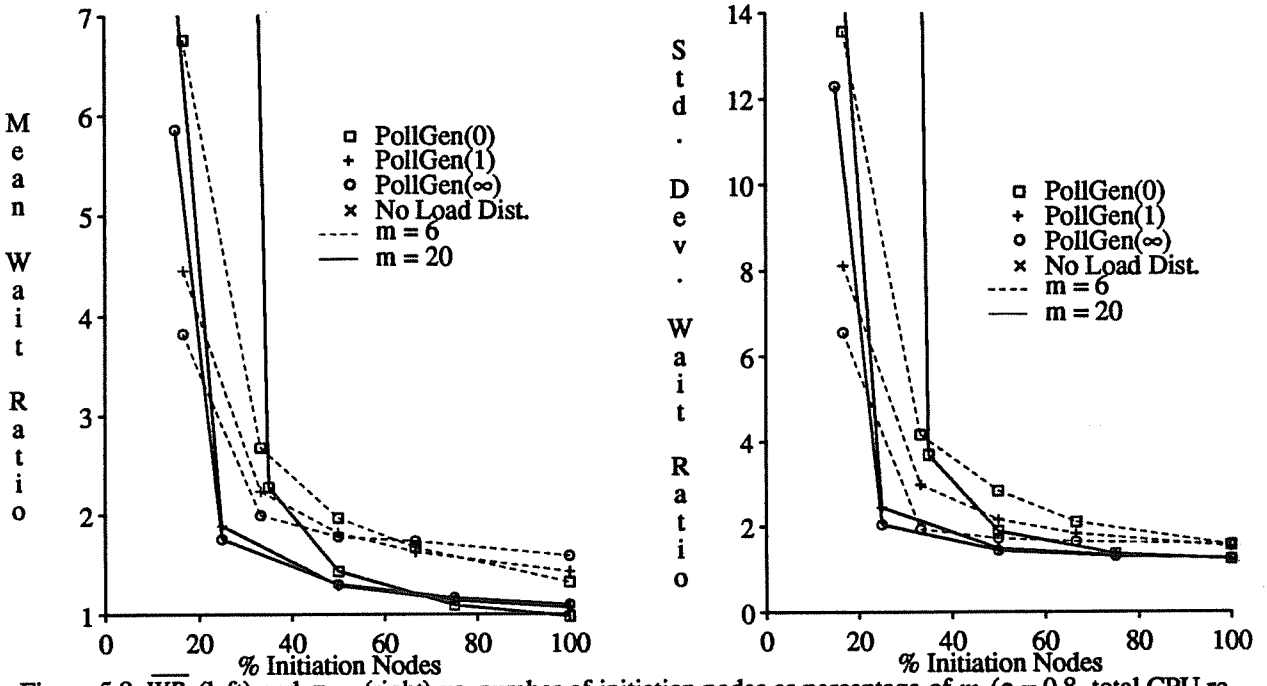

Figure 5.8 $\overline{WR}$ (left) and σ_{WR} (right) vs. number of initiation nodes as percentage of m ($\rho = 0.8$, total CPU requirement to migrate a process of average size = 14.4%)

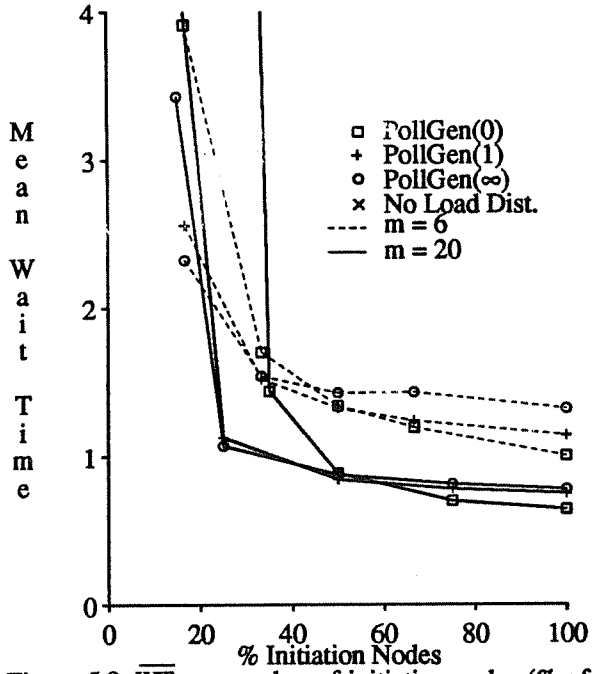

Figure 5.9 $\overline{WT}$ vs. number of initiation nodes (% of m) ($\rho = 0.8$, CPU requirement for mig. = 14.4%)

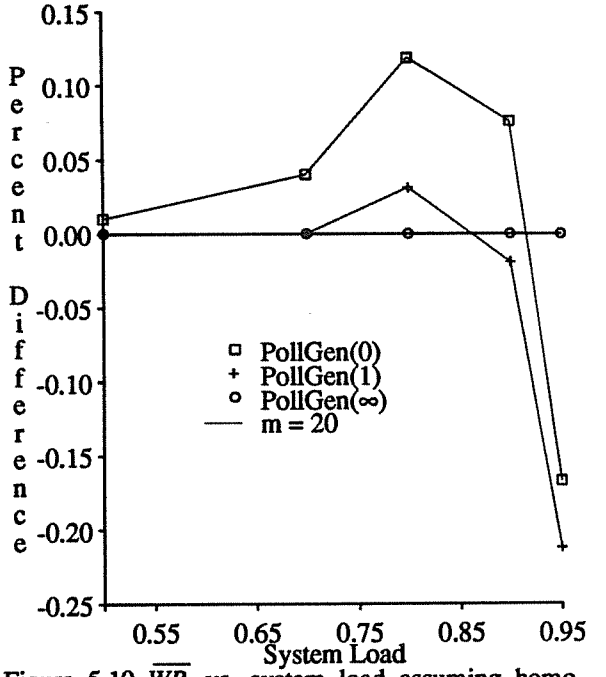

Figure 5.10 $\overline{WR}$ vs. system load assuming homogeneous initiation rates ($m = 20$, CPU requirement for mig. = 2.4%)

6. Conclusions

In [Krueg86], it was shown that, in terms of the LB objective, the LB strategy has a significant performance advantage over the LS strategy when the resource requirements of load distributing are negligible. In addition, in terms of the *LS* objective, the LB strategy has a significant advantage when process service demands are hyperexponentially distributed. These performance advantages were shown to increase with increasing inhomogeneity in process initiation rates, system load and coefficient of variation in process service demands. In this paper, we have shown that the resource requirements of the LB and LS strategies have a significant effect on these performance advantages, and that the nature of this effect depends on the pattern of process initiation rates among nodes. When there is a single initiation node, cost *amplifies* the LB advantages. Answering the question posed by the title of this paper, under such a workload, the LB strategy best meets both the LB *and* the LS objectives. However, when initiation rates are homogeneous, cost has a *degrading* effect on the LB advantage, until the LB strategy becomes a disadvantage at high cost. Under such a workload, the strategy that best meets the LB objective varies with increasing cost, from LB, through LS with anticipatory migrations, continuing through LS without anticipatory migrations, culminating with no load distributing at very high cost. The LS objective is best met, under such a workload, by the LS strategy unless cost is very high, when it is best met without load distributing. Transitions in the strategy that best meets the LB or LS objective also occur with changes in system load or the level of inhomogeneity in process initiation rates. To summarize, while there is an essential distinction between the LB and LS objectives, The LB and LS strategies do not form a dichotomy, but are simply two points on a continuum of strategies that can be used to meet either the LB or LS objective. The strategy that best meets the LB or LS objective is not determined solely by the objective, but by the objective *together* with the system workload and resource availability.

Many of the factors that determine the strategy that best meets a given objective are dynamic. These factors include system load, the distributions of process service demands and physical process sizes, the utilization of the communication device for purposes other than load distributing, and the number of nodes participating in the system. In addition, as shown in [Mutka87], the pattern of process initiation rates across nodes can be expected to be highly variable, with a high level of inhomogeneity in initiation rates being the rule. We conclude that to be effective at meeting either the LB or the LS objective over the wide range of conditions

occurring within a distributed system, a load distributing algorithm must adapt to its environment. Our current research [Krueg86a] focuses on the design of such algorithms. An LB algorithm that adapts to communication device cost is studied in [Krueg84].

When a performance objective is based on more than one performance index, such as the LB objective, the point at which that objective is best met in a given environment may not be clear. Improvement of individual performance indices may be mutually exclusive, and improvement in one may have to be weighed against degradation in another.

Finally, we note that the performance resulting from either the LB or LS strategy is very sensitive to the CPU requirements of negotiation and migration. Efficient implementation of these load distributing operations is important in achieving maximum performance.

7. Appendix: Negotiation, Migration and Unshared Rates in M/M/m-like Systems

In this section, we derive the negotiation and migration rates necessary to keep an M/M/m-like system in a balanced or shared state. In the interest of generality, we calculate the rates at which negotiation sessions are initiated, rather than the rates at which individual negotiation messages are generated. In addition, we calculate the rates at which individual nodes enter the unshared state, denoted $\hat{UNS}$, which is the sum of the rate at which jobs arrive at busy nodes while some nodes are idle and the rate at which nodes become idle while processes wait for service. Since LB and LS in an M/M/m-like system both result in work-conserving systems regardless of the local scheduling discipline, the results of this section are valid for any local scheduling discipline allowed within the M/M/m-like model. Notation used in this section is summarized in table 2.1.

7.1. Load Balancing

To assure that an M/M/m-like system is always in a balanced state, negotiation must be initiated whenever either of the following events occurs:

E1 a process initiates at a node that is busy

E2 a process completes execution

These events can be further subdivided into occurrences that spark successful negotiation and result in process migrations and those that initiate unsuccessful negotiations:

E1s a process initiates at a node having greater than the mean number of processes
E1u a process initiates at a node having the arithmetic mean or fewer processes
E2s a process completes at a node having less than the mean number of processes
E2u a process completes at a node having the arithmetic mean or more processes

The negotiation rate is the sum of the rates of occurrence of events E1 and E2, denoted $E\hat{1}$ and $E\hat{2}$, while the migration rate is the sum of $E\hat{1}s$ and $E\hat{2}s$.

Lower Bound: Homogeneous Arrival Rates

The rate at which processes initiate at busy nodes when initiation rates are homogeneous is:

$$E\hat{1} = \lambda_i E(number\ of\ busy\ nodes) = \lambda_i \left[\sum_{n=1}^{m-1} np_n + m \sum_{n=m}^{\infty} p_n \right]$$

For $\rho \leq 1$, the rate at which processes complete execution is equal to the initiation rate: $E\hat{2} = \lambda$. Since the number of nodes having more than the mean number of processes is $n \bmod m$:

$$E\hat{1}s = \lambda_i E(number\ of\ nodes > mean) = \lambda_i \sum_{n=0}^{\infty} (n \bmod m) p_n$$

When $n > m$, it is possible for event E2s to occur. The number of nodes with less than the mean number of processes is $m-(n \bmod m)$, so:

$$E\hat{2}s = \mu_i E(number\ of\ nodes > 0 \text{ and } < mean) = \mu_i \sum_{n=m+1}^{\infty} (m-(n \bmod m)) p_n$$

Under load balancing, processes can arrive at busy nodes while nodes are idle only when $1 \leq n < m$, and nodes can become idle while processes wait for service only when $m < n < 2m$:

$$U\hat{N}S = \lambda_i \sum_{n=1}^{m-1} np_n + \mu_i \sum_{n=m+1}^{2m-1} (2m-n) p_n$$

Upper Bound: Single Arrival Node

From the state diagram of an M/M/m queue (see [Trive82], pg. 374), the probability that the initiation node has greater than the mean number of processes, given that the system contains n processes, can be seen to be:

$$O_n = \begin{cases} 0 & \text{if } n \bmod m = 0 \\ \dfrac{\lambda p_{n-1} + \left\lceil \dfrac{n \bmod m}{(n \bmod m)+1} \right\rceil (n+1)\mu_i O_{n+1} p_{n+1}}{\lambda p_{n-1} + (n+1)\mu_i p_{n+1}} & \text{if } 0 < n < m \\ \dfrac{\lambda p_{n-1} + \left\lceil \dfrac{n \bmod m}{(n \bmod m)+1} \right\rceil m\mu_i O_{n+1} p_{n+1}}{\lambda p_{n-1} + m\mu_i p_{n+1}} & \text{if } n > m \text{ and } n \bmod m <> 0 \end{cases}$$

From this:

$$E\hat{1} = \lambda\left[\sum_{n=1}^{m-1} O_n p_n + \sum_{n=m}^{\infty} p_n\right] \qquad E\hat{1}s = \lambda\sum_{n=1}^{\infty} O_n p_n$$

Again, since processes can arrive at busy nodes while nodes are idle only when $1 \le n < m$, and nodes can become idle while processes wait for service only when $m<n<2m$:

$$\hat{UNS} = \lambda\sum_{n=1}^{m-1} O_n p_n + \mu_i \sum_{n=m+1}^{2m-1} (2m-n)p_n$$

$E\hat{2}$, $E\hat{2}s$, and $E\hat{2}u$ are identical to those in a system having homogeneous initiation rates, since these events depend on process completions rather than initiations, and, under load balancing, the distribution of processes among nodes is not dependent on the homogeneity of initiation rates.

7.2. Load Sharing

An algorithm that guarantees that an M/M/m-like system is always in a shared state initiates negotiation whenever either of the following events occur:

- E1 a process initiates at a node that is busy
- E2 a node becomes idle

These events can be subdivided into those that result in successful negotiations and those that do not:

- E1s a process initiates at a busy node while some node is idle
- E1u a process initiates at a busy node while no node is idle
- E2u A node becomes idle when no node has more than one process
- E2s A node becomes idle while at least two processes reside at another node

For load sharing, $\hat{UNS}$ is equal to the migration rate ($E\hat{1}s + E\hat{2}s$).

Lower Bound: Homogeneous Arrival Rates

Our derivation of the migration rate when process initiation rates are homogeneous is similar to that of Livny [Livny83], though we find a tighter lower bound. The rate at which processes initiate at busy nodes is:

$$E\hat{1} = \lambda_i E(\textit{number of busy nodes}) = \lambda_i \left[\sum_{n=1}^{m-1} np_n + m \sum_{n=m}^{\infty} p_n \right]$$

Separating this, $E\hat{1}s = \lambda_i \sum_{n=1}^{m-1} np_n$ and $E\hat{1}u = \lambda_i m \sum_{n=m}^{\infty} p_n$. Under load sharing, event E2u occurs whenever a process completes at a node when $n \leq m$, so $E\hat{2}u = \mu_i \sum_{n=0}^{m} np_n$. Conversely, E2s can occur only when $n > m$. Under the assumption that, when $n > m$, all assignments of the $n-m$ 'excess' processes to nodes are equally likely, the probability that all $n-m$ processes are at nodes other than a given node is $((m-1)/m)^{n-m}$, and $E\hat{2}s = \mu_i m \sum_{n=m+1}^{\infty} \left[\frac{m-1}{m} \right]^{n-m} p_n$. As noted in section 3, simulations show that, rather than being equally likely, there is a tendency toward assignments of processes to nodes that are more unbalanced. While this tendency increases $E\hat{2}s$, simulation results show close agreement.

Upper Bound: Single Arrival Node

From the state diagram of an M/M/m queue, the probability that the initiation node is busy, given that there are n processes in the system, can be seen to be:

$$B_n = \begin{cases} 0 & \text{if } n = 0 \\ \dfrac{\lambda p_{n-1} + \left[\dfrac{n \bmod m}{(n \bmod m)+1} \right] (n+1)\mu_i B_{n+1} p_{n+1}}{\lambda p_{n-1} + (n+1)\mu_i p_{n+1}} & \text{if } n < m \\ 1 & \text{if } n \geq m \end{cases}$$

From this, $E\hat{1}s = \lambda \sum_{n=1}^{m-1} B_n p_n$. Since, under load sharing, no node is idle if $n \geq m$, so $E\hat{1}u = \lambda \sum_{n=m}^{\infty} p_n$. When all processes initiate at the same node, load sharing allows no other node to have more than one process. The rate at which these other nodes become idle when at least two processes reside at the initiation node is $E\hat{2}s = \mu_i (m-1) \sum_{n=m+1}^{\infty} p_n$. When $n \leq m$, no node has more than one process, so $E\hat{2}u = \mu_i \sum_{n=1}^{m} np_n$.

References

[Bryan81] R. M. Bryant and R. A. Finkel, "A Stable Distributed Scheduling Algorithm," *Proc. Second International Conference on Distributed Computing Systems*, pp. 314-323 (April 1981).

[Eager86] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering SE-12*, 5, pp. 662-675 (May 1986).

[Eager86a] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Dynamic Load Sharing," *Performance Evaluation 6*, 1, pp. 53-68 (March 1986).

[Klein76] L. Kleinrock, *Queuing Systems: Volume 2, Computer Applications*, John Wiley & Sons (1976).

[Krueg84] P. Krueger and R. Finkel, "An Adaptive Load Balancing Algorithm for a Multicomputer," Technical Report 539, University of Wisconsin–Madison, Dept. of Computer Sciences (April 1984).

[Krueg86a] P. Krueger, "Adaptivity in Load Distributing Algorithms: A Preliminary Report," Preliminary Report, University of Wisconsin–Madison, Dept. of Computer Sciences (August 1986).

[Krueg86] P. Krueger and M. Livny, "Load Balancing, Load Sharing and Performance in Distributed Systems," submitted for publication, University of Wisconsin–Madison, Dept. of Computer Sciences (December 1986).

[Laven83] S. S. Lavenberg, *Computer Performance Modeling Handbook*, Academic Press (1983).

[Livny82] M. Livny and M. Melman, "Load balancing in homogeneous broadcast distributed systems," *Computer Network Performance Symposium*, pp. 47-55 (April 1982).

[Livny83] M. Livny, The Study of Load Balancing Algorithms for Decentralized Distributed Processing Systems, PhD Thesis, Weizmann Institute of Science, Rehovot, Israel (available as Technical Report 570, University of Wisconsin–Madison Computer Sciences) (August 1983).

[Mutka87] M. Mutka and M. Livny, "Profiling Workstations' Available Capacity for Remote Execution," submitted for publication, University of Wisconsin–Madison, Dept. of Computer Sciences (April 1987).

[Trive82] K. S. Trivedi, *Probability & Statistics with Reliability, Queuing and Computer Science Applications*, Prentice-Hall (1982).